Université du Québec à Chicoutimi **Session : h2020**
Département d'Informatique et de Mathématiques **Pr.: A. Bouzouane**

8PRO114: Programmation Orientée Objet

Mini-projet (peut être réalisé en binôme)
À remettre le 15 Avril 2020[1]

---

1. But

Permettre aux étudiants de se familiariser avec:

- **La réutilisation avec l'héritage multiple;**
- **La généricité par l'application de la programmation générique;**
- **La robustesse à travers la gestion des exceptions;**
- **Le «refactoring» pour faciliter la maintenance;**
- **Le multithreading pour paralléliser certains services.**

2. Travail à effectuer

IL s'agit de compléter l'application «LigueSoccerApp» réalisée dans le TP2, par l'ajout de nouvelles classes pour un jeu de négociation de transfert de joueurs autonomes. Le transfert sera basé sur une simulation d'une négociation en traitement simultanée, entre un négociateur du club Vendeur (*negoVendeur)* et un autre du club Acheteur (*negoAcheteur*), à l'aide du mécanisme de multithreading. Vous pouvez proposer une autre stratégie outre que celle qui est décrite ci-dessous[2].

L'objectif visé est de réaliser une transaction commerciale selon un certain nombre de contraintes. L'exemple de contraintes qu'on peut prendre en considération, concernent les montants des transferts (vente et achat) fixés par les clubs, et aussi la date limite du transfert (mercatos) imposée

---

[1] À déposer dans le dossier de remise sur le site moodle de cours.

[2] **Décrire la stratégie dans un fichier (.doc ou (.txt).**

par la ligue. Cette date sera matérialisée par la **durée de la simulation**. Ces contraintes seront prises par les négociateurs de chaque club, dans le but de maximiser (resp. de minimiser) les profits (les coûts). Par exemple, dans le cas d'un *negoVendeur*, son but est de maximiser le montant de la transaction. Pour cela, sa stratégie de négociation consiste à faire une première proposition selon ***un montant maximal désiré*** *($9M selon l'exemple de la figure de la page 4),* et à **diminuer** ce prix en fonction du temps jusqu'à recevoir une offre acceptable respectant le **montant minimal** fixé (*p.e., $4M d'après la figure*). Si la durée de la négociation (*p.e., 8 secondes, c.f. figure*) s'est achevée et qu'il n'a pas encore reçu une offre supérieure au prix minimal, alors le transfert du joueur est échoué. La classe *negoVendeur* doit implanter un **thread de négociation** et doit être caractérisée au moins par les informations suivantes :

- ***montant désiré*** *du transfert;*
- ***montant minimal acceptable*** *de la vente (transfert);*
- ***durée*[3] *de négociation*** *peut être en millisecondes;*
- ***representantClub*** *de type Club.*

L'idée du jeu, est d'échanger des messages d'offres et de contre-offres entre deux négociateurs à propos d'un montant de transfert d'un joueur donné. La **variation** du montant peut être simulée par une des fonctions linéaires suivantes:

1) Montant désiré= Montant désiré – temps écoulé
2) Montant désiré= Montant désiré / temps écoulé

Le temps écoulé représente le temps depuis le début de la transaction et peut être calculé de la manière suivante :

Temps écoulé= temps courant – temps au début de la transaction.

Le ***temps au début de la transaction*** représente l'instant où la négociation a débutée, et le ***temps courant*** représente le moment où le négociateur décide de faire une contre-offre, suite à une proposition reçue. Ils peuvent être estimés en millisecondes, et calculés en moyennant une lecture du temps (horloge) de la machine.

[3] *La durée du transfert sera la même que celle du négociateur pour l'achat.*

À partir de là, on doit faire une lecture du temps machine au début de la transaction. Lors de la réception d'un message concernant une offre, pour le traiter et répondre, on doit aussi lire le temps machine avant de générer un nouveau message sur une nouvelle offre basée sur la formulation précédente et le message envoyé par l'autre négociateur. Par exemple, le *negoVendeur* décide de se départir de son joueur, si l'offre concernant le nouveau montant proposé par le *negoAcheteur* qui est contenu de son message, est comprise entre le montant désiré par le *negoVendeur* et le montant minimal acceptable et imposé par son club, et que la durée de la négociation est terminée (*la fin du mercatos*). Le comportement du *negoVendeur* peut se traduire à travers les règles suivantes. Vous pouvez aussi introduire d'autres règles de comportements du vendeur :

Si *le montant proposé par le negoAcheteur est inférieur au plus bas montant acceptable et que la durée n'est pas terminée*

Alors

Rejeter la proposition et faire une contre proposition (une offre) basée sur la formule d'estimation du nouveau montant

Si *le montant proposé par le negoAcheteur* ***est compris*** *entre le montant désiré et le plus bas montant acceptable et que la durée est terminée*

Alors

Accepter la proposition en envoyant un message à l'acheteur et instancier la classe contrat d'engagement.

L'instanciation du **Contrat** d'engagement doit mettre fin à la négociation en mettant à jour les informations appropriées.

Dans le cas d'un *negoAcheteur*, sa stratégie implantée aussi à l'aide d'un **thread**, est de minimiser ses coûts en agissant dans le sens inverse du vendeur. Cela consiste à faire une première proposition sur un **montant minimal désiré** *(p.e. $1M d'après l'exemple de la figure),* et à **augmenter** ce montant en fonction du temps jusqu'à recevoir une proposition acceptable respectant le prix **maximal** *(p.e. $7M)* qu'il a fixé au début de la transaction. Si la durée de la négociation est achevée et qu'il n'a pas reçu d'offres inférieures au prix maximal, alors le transfert du joueur est échoué. La classe *negoAcheteur* sera caractérisée, entre autres:

- ***montant désiré*** *d'achat désiré;*
- ***montant maximal*** *acceptable du transfert;*

- ***durée de négociation*** *peut être en millisecondes;*
- ***représentantClub*** *de type Club.*

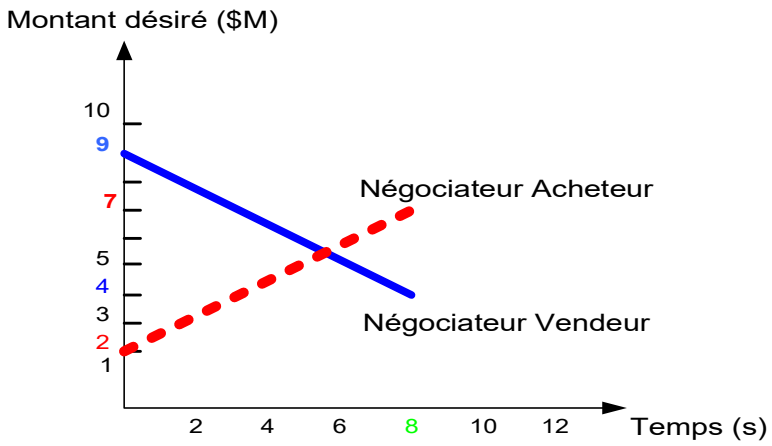
La variation du montant peut être calculée en utilisant la formulation précédente, avec cette modification :

Montant désiré= Montant désiré + temps écoulé

Voici un exemple de règle reflétant le comportement de l'acheteur:

Si *La durée s'achève et*
L'offre de prix proposé par le vendeur est comprise entre le montant désiré et le plus haut montant acceptable
Alors
envoyer une proposition d'achat du joueur en générant un message

On peut résumer cette stratégie de négociation par la figure suivante :



À partir de là, on peut remarquer que les négociateurs peuvent ou ne peuvent pas arriver à une entente. Vous devez créer une classe *Negociation* supportant les méthodes (protocoles) de négociation suivantes:

1. ***ProposerOffre ()*** *: proposer une nouvelle offre sur le montant de transfert d'un joueur.*
2. ***AccepterOffre()*** *: accepter l'offre proposée.*
3. ***RejeterOffre()****: rejeter l'offre proposée.*

Ces méthodes consistent à créer des messages qui épouseront la structure suivante :

- ***l'émetteur du message*** : *le négociateur qui émet le message*
- ***le sujet du message*** : *offre, acceptation, rejet,…*
- ***montant courant*** *du transfert*

Ces messages peuvent être déposés dans une **file de messages** associée à chacune des classes *negoVendeur* et *negoAcheteur*, qui à la réception d'un message, le négociateur réagira en proposant une contre-offre via un autre message.

L'application doit:

3.1 permettre la création d'une transaction de négociation entre négociateurs, en affichant la liste des messages échangés entre eux. Vous devez créer une super classe **Negociateur** pour abstraire les classes *negoVendeur* et *negoAcheteur*. **La visualisation du graphe de la stratégie de négociation est facultative mais il y aura un bonus important pour la note finale. Le recours à des APIs (librairies) est non limité**.

3.2 créez ou modifiez une ou plusieurs classes pour mettre en place le principe du polymorphisme à l'aide des fonctions virtuelles pures.

3.3 prendre en charge au moins un exemple sur la généricité. Si une fonction est implémentée plusieurs fois, alors vous devez utiliser le mécanisme de patron de fonction pour avoir une seule version d'une telle fonction. Je vous laisse le choix de l'exemple d'implantation de ce concept pour ***améliorer la qualité*** de l'application. Vous pouvez aussi créer ou modifier une ou plusieurs classes afin de créer un template de classe pour illustrer ce concept.

3.4 prendre en charge au moins un exemple sur l'héritage multiple. Vous devez créer ou modifier une ou plusieurs classes pour implanter un exemple d'héritage multiple, et ainsi ***maximiser la réutilisation*** des composantes (classes ou fonctions).

3.5 permettre un service de sauvegarde et de restauration des objets à partir de fichiers séquentiels. Ce service peut être implanté à travers une ou plusieurs classes qui feront office d'interface entre

l'application et les fichiers impliqués. Il permettra ainsi d'éviter d'instancier à chaque exécution les classes persistantes, telles que la classe Joueur, Equipe, etc. Il est préférable d'associer pour chaque classe persistante, un fichier afin de faciliter la reconstruction des objets. Le recours à d'autres mode de stockage, telles que les bases de données ou autres sont les bienvenues.

3.6 doit être robuste en prenant en compte l'ensemble des **exceptions d'entrées/sorties.**

3.7 respecter les principes de la programmation objet : abstraction, encapsulation, héritage, polymorphisme, et ainsi que les règles du refactoring.

3.8 <u>être opérationnelle sur l'ensemble des services et des fonctionnalités demandés depuis le début des travaux pratiques (TP1, TP2) dans le cadre de ce projet final.</u>

NB : <u>Un bonus sera attribué au travail de qualité : convivialité, architecture, ajout de nouvelles fonctionnalités non demandées, etc</u>.

4. Bien livrable

4.1 L'exécutable ainsi quelques exemples de copies d'écrans (résultats d'exécutions) ;

4.2 le code source documenté, validé de l'application et les fichiers (.h et .cpp)

4.3 Une page de garde contenant votre nom, prénom et le code permanent (.txt)

4.4 S'il y a lieu, un fichier (.doc ou .txt) contenant la description de la stratégie de négociation que vous proposez, outre que celle qui est proposée.

<u>L'ensemble doit être dans votre compte.</u>

Bon succès.